**1. Canny Edge Detection**

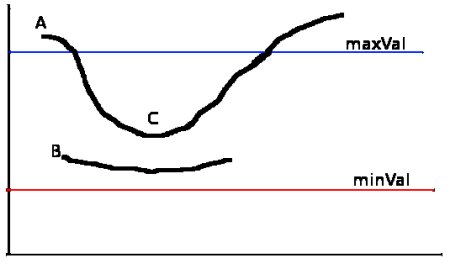
**Canny Edge Detection** là một thuật toán trong xử lý ảnh được phát triển bởi John F. Canny vào năm 1986 để tìm ra các biên cạnh trong ảnh. Thuật toán này được thiết kế để giảm thiểu các sai số và tối đa hóa độ chính xác của các biên cạnh được phát hiện. Canny Edge Detection bao gồm các bước chính sau:

1. Làm mịn ảnh (Smoothing): Trước tiên, ảnh gốc được làm mịn bằng cách sử dụng một bộ lọc Gaussian 5x5 để giảm nhiễu và loại bỏ các chi tiết không mong muốn từ ảnh.

2. Tính toán đạo hàm (Gradient Calculation): Sau đó, gradient của ảnh được tính toán bằng cách sử dụng các bộ lọc gradient như Sobel để xác định hướng và độ lớn của biên cạnh tại mỗi điểm ảnh.

3. Tìm ngưỡng (Thresholding): Gradient được sử dụng để xác định các điểm ảnh có độ lớn gradient cao, có thể liên quan đến các biên cạnh. Sau đó, hai ngưỡng được sử dụng - ngưỡng dưới và ngưỡng trên. Các điểm ảnh có độ lớn gradient lớn hơn ngưỡng trên được coi là điểm ảnh của biên cạnh chắc chắn. Các điểm ảnh có độ lớn gradient thấp hơn ngưỡng dưới được loại bỏ. Các điểm ảnh có độ lớn gradient nằm giữa ngưỡng dưới và ngưỡng trên sẽ được xem xét theo nguyên tắc hysteresis (ngưỡng trên và dưới kết hợp). Nếu điểm ảnh này kết nối với một điểm ảnh được xác định là biên cạnh chắc chắn, nó sẽ được coi là điểm ảnh của biên cạnh.

4. Loại bỏ các điểm ảnh không cần thiết (Edge Tracking by Hysteresis): Bước này liên quan đến việc loại bỏ các điểm ảnh được đánh giá là không chắc chắn là biên cạnh. Các điểm ảnh này có thể xuất hiện do nhiễu hoặc các chi tiết không mong muốn. Canny Edge Detection sử dụng hysteresis để loại bỏ các điểm ảnh này.



Kết quả của thuật toán Canny Edge Detection là một ảnh nhị phân, trong đó chỉ có hai giá trị: 0 (đen) hoặc 255 (trắng), thể hiện các điểm ảnh không phải là biên cạnh hoặc là biên cạnh. Các biên cạnh được phát hiện thông qua thuật toán này thường rất sắc nét và chính xác.

Code:

gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
edges\_image = cv2.Canny(gray\_image, 100, 200, apertureSize=3, L2gradient=**True**)

Doc: <https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html>

**2. Hough Line Transform**

Phương pháp biến đổi Hough (**Hough Transform**) được phát triển bởi Paul Hough vào năm 1962, ban đầu để nhận diện các đoạn thẳng trong hình ảnh. Hough Line Transform (Biến đổi đường thẳng Hough) là một ứng dụng cụ thể của phương pháp Hough để tìm các đoạn thẳng trong ảnh. Cách Hoạt Động:

1. Điểm và Phép Biến Đổi Hough:

Mỗi điểm ảnh trong ảnh có thể được biểu diễn dưới dạng một đường thẳng trong không gian tham số Hough.

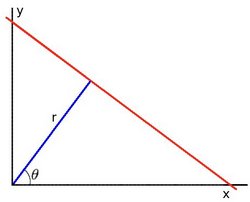
Đối với một điểm ảnh có tọa độ (x,y) trong ảnh, đường thẳng đó có thể được biểu diễn bằng phương trình y=mx+b trong không gian tham số Hough, với m là độ dốc của đường thẳng và b là sai số chéo.

1. Biểu Diễn Đường Thẳng trong không gian Hough:

Đối với mỗi điểm ảnh (x,y) trong ảnh, ta có một đường thẳng trong không gian Hough được biểu diễn bằng phương trình .

+ Với hệ tọa độ Cartesian: tham số là (m, b)

+ Với hệ tọa độ Polar: tham số là (r, θ)



Phương trình đường thẳng trong hệ Polar:

1. Biến Đổi Hough:

Để tìm các đoạn thẳng trong ảnh, chúng ta duyệt qua tất cả các điểm ảnh và biểu diễn các đường thẳng tương ứng trong không gian Hough.

Mỗi điểm (m,b) trong không gian Hough đại diện cho một đường thẳng trong ảnh. Càng nhiều điểm (m, b) trùng nhau, càng có khả năng rằng có một đoạn thẳng tương ứng trong ảnh.

1. Tìm Các Đoạn Thẳng:

Trong không gian Hough, các điểm (m, b) tập trung thành các điểm nhiều nhất chính là các đoạn thẳng trong ảnh gốc.

Để tìm các đoạn thẳng, ta có thể áp dụng một ngưỡng để chọn các điểm nhiều nhất và sau đó chuyển các điểm này trở lại không gian ảnh để nhận diện các đoạn thẳng tương ứng.

1. Ưu Điểm và Nhược Điểm:

Ưu Điểm: Phương pháp Hough Line Transform có thể nhận diện các đoạn thẳng trong ảnh mà không bị ảnh hưởng bởi nhiễu hoặc các đoạn thẳng không liên tục.

Nhược Điểm: Tuy nhiên, việc tính toán trong không gian Hough đòi hỏi nhiều tài nguyên, đặc biệt là với ảnh có độ phức tạp cao, và cũng có thể tốn nhiều thời gian tính toán.

Với mỗi pixel, đường thẳng đi qua nó đè lên càng nhiều điểm thì càng có khả năng đó là 1 đường thẳng (dùng ngưỡng để xác định). VD threshold=100 nghĩa là đè lên 100 điểm trở lên thì được tính là 1 đường thẳng

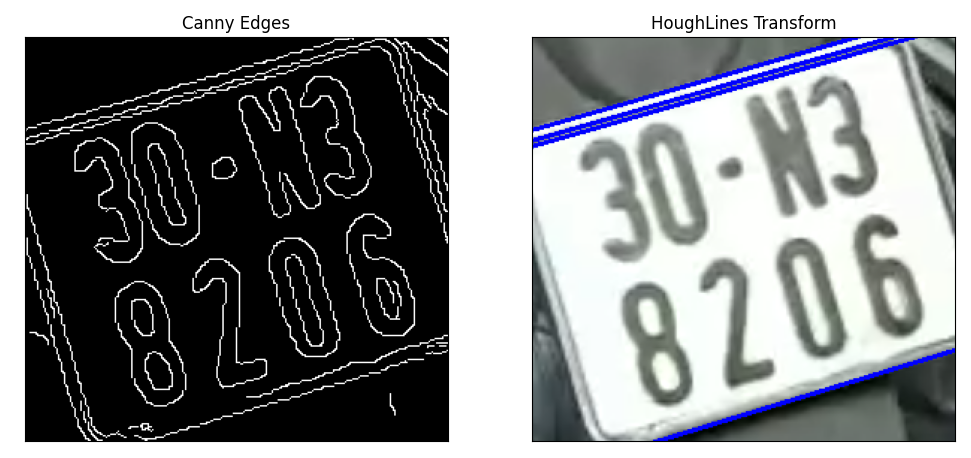
Code:

lines = cv2.HoughLines(edges\_image, 1, np.pi / 180, threshold=100)

Doc: <https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html>

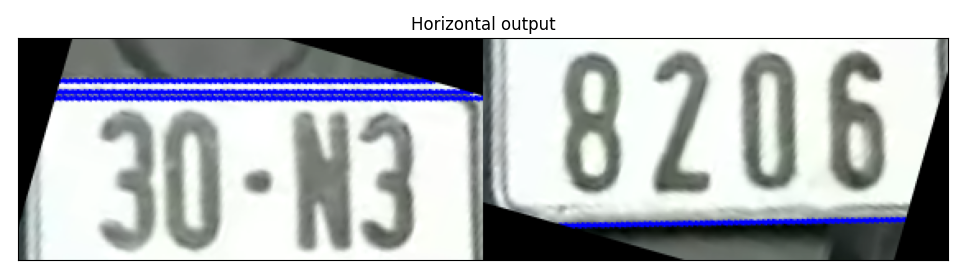
**3. Tìm góc xoay phù hợp**

Sau khi đã vẽ được các đường thẳng, ta có hình như sau:



Bây giờ việc cần làm là lấy ra 1 đoạn thẳng để so sánh với khung ảnh để tính ra góc xoay. Mục tiêu là làm cho biển số xe **nằm ngang** để thuận tiện có mô hình OCR có thể đọc.

Phương pháp đưa ra là lấy ra đoạn thẳng **dài nhất** trong hình, sau đó tính góc tạo bởi đoạn thẳng đó và đường biên của ảnh, đó chính là **góc cần phải xoay**. Kết quả như sau:



Code:

**if** filtered\_lines:  
 *# Lựa chọn đoạn thẳng dài nhất và thẳng nhất* longest\_line = max(filtered\_lines, key=**lambda** x: np.linalg.norm(np.array(x[0]) - np.array(x[1])))  
 x1, y1 = longest\_line[0]  
 x2, y2 = longest\_line[1]  
  
 *# Tính góc xoay của đường thẳng* rotation\_angle = (np.arctan2(y2 - y1, x2 - x1) \* 180 / np.pi)  
  
 *# Xoay lại ảnh để biển số xe nằm ngang* height, width = image.shape[:2]  
 center = (width // 2, height // 2)  
 rotation\_matrix = cv2.getRotationMatrix2D(center, rotation\_angle, 1)  
 rotated\_image = cv2.warpAffine(image, rotation\_matrix, (width, height))